Comparing SOAP Performance for Various Encodings, Protocols, and Connections

Jaakko Kangasharju, Sasu Tarkoma, and Kimmo Raatikainen

Helsinki Institute for Information Technology
PO Box 9800, 02015 HUT, Finland
Tel: +358 50 384 1518, Fax: +358 9 694 9768
{jaakko.kangasharju,sasu.tarkoma,kimmo.raatikainen}@hiit.fi

Abstract. SOAP is rapidly gaining popularity as the Web service protocol. At the same time, small mobile devices with wireless access, in particular to the Internet, are becoming more prevalent. At first look, it would seem that SOAP as a protocol consumes quite a lot of network bandwidth and processor time. Therefore its suitability for small devices and wireless links needs to be evaluated. This paper presents two optimizations that can be applied to typical uses of SOAP, message compression and persistent connections, and measures their performance in some common situations. Asynchronous messaging with SOAP is also treated briefly. The measurements indicate that a suitable compression scheme can save bandwidth substantially, and that the protocols underlying the typical use of SOAP can be improved considerably in the presence of unreliable high-latency networks.

Keywords: Measurement Of Wireless And Mobile Systems, Mobile And Wireless Applications, Web Services Over Mobile And Wireless Networks

1 Introduction

Web services are a rising phenomenon in the network service world, and SOAP is the protocol of Web services. One of SOAP's benefits is that, being XML, it is more human-readable than binary protocols and therefore easier to debug. Another often-mentioned benefit is the use of the ubiquitous HTTP as a tunneling protocol, which e.g. allows SOAP messages to penetrate firewalls.

A visible trend in future communications is a large increase in the use of mobile terminals. These include small devices with limited processing power and a wireless connection to larger networks. Any future ubiquitous communication protocol should preferably be usable even on these devices. The main difficulties in wireless communication compared to fixed links are significantly lower bandwidth, which limits the amount of data that can be sent, and higher latency, which limits the number of round trips a protocol can effectively make within a communication.

The most common version of SOAP currently in use is SOAP 1.1 [9] while SOAP 1.2 [10] is being prepared by the World Wide Web Consortium. The most

M. Conti et al. (Eds.): PWC 2003, LNCS 2775, pp. 397–406, 2003.

common underlying protocol used for SOAP messages is HTTP 1.0 [4]. SOAP 1.1 requires messages to be encoded as XML documents, which are not designed to be space-efficient. While SOAP messages are fundamentally one-way, a common use of SOAP is for synchronous RPC, which requires the application to wait for a network round trip for each message sent. Additionally, HTTP 1.0 supports only a single exchange per TCP connection, so each invocation requires an additional round trip to first establish a connection. Thus typical current use of SOAP would seem to accentuate the main problems in wireless communication.

The size of XML documents can possibly be overcome by compressing them in some manner. Unfortunately, SOAP messages are often quite small, so typical compression algorithms, such as the popular Gzip [5] supported by many HTTP clients and servers, will not manage very well. There are indications that a binary representation of XML, such as WAP Binary XML [11], could do better than traditional compression in this case [7].

The latency problems caused by the need to reopen connections when using HTTP 1.0 could be solved simply by moving to HTTP 1.1 [6], which supports persistent connections. The SOAP programming framework could also offer asynchronous operations, which could eliminate the effects of high latency for client applications not needing synchronicity. Asynchronous operations could, for example, be implemented on top of HTTP's synchronous request-response, but it would also be possible to move to full asynchronicity even at the transfer protocol level.

The problems of TCP over wireless links are well known [1] and several improvements to TCP have been proposed [3]. Our work here is more concerned on the impact of different TCP usage patterns on application performance, rather than gaining improvements by modifying TCP itself. Apart from the compression approaches mentioned above, a good XML-specific compressor is XMill [8], which is, however, better suited to large documents. The MPEG-7 standard [2] also includes a specification for a binary XML format.

In this paper we examine performance implications of alternatives to the HTTP protocol binding for SOAP. The alternatives include use of plain TCP, Gzip compression of XML documents, persistent TCP connections, and cache-based tokenization of SOAP message items. The rest of the paper is organized as follows: In Sect. 2 we describe what tests were run and in which environments, in Sect. 3 we present our results, and Sect. 4 gives the conclusions based on the results, as well as plans for extending this work in the future.

2 Test Description

A test system with easily exchangeable protocol implementations was set up to measure the effects of the two above-mentioned optimizations, compression of messages and persistent connections. Tests were run on several different connections to see how link quality degradation affects protocol performance. These tests were intended to identify the benefits and drawbacks of different optimizations.

Table 1. The message exchange scenarios used in the tests

Scenario	Description
Deploy	The client sends a 391-byte XML document to the server in a SOAP message (total size 655 bytes) and receives a 31-byte XML document back in a SOAP message (total size 296 bytes)
Stress	The client sends, as quickly as possible, 17472[a] times a 258-byte XML document in a SOAP message (total size 652 bytes), each message receiving a 195-byte XML document back in a SOAP message (total size 464 bytes)

[a] 17472 equals $2 \times 24 \times 364$; the test system was based on a calendaring application, so the Stress scenario measures the cost of filling every half-hour of a year (minus a day)

The SOAP framework used in testing was Apache Axis 1.0 running on Debian GNU/Linux 3.0 with Linux version 2.4.18. The machines used for testing were one desktop PC and two laptop PCs with the desktop PC having a fixed network connection and the laptop PCs having a choice of fixed and wireless connections. The desktop PC had a 1333 MHz AMD Athlon processor and 512 MB of main memory. The laptop PCs's model was HP Omnibook 500 with a 500 MHz Intel Pentium III processor and 512 MB of main memory.

During testing the machines were in normal multi-user operation, but with no other significant computation proceeding at the same time. Apache Axis is implemented in the Java programming language; the Java versions used were Java 2 SDK 1.4.1 on the desktop and Java 2 SDK 1.3.1 on the laptops, both from Sun Microsystems. All network traffic generated by the tests was captured with the Ethereal network traffic analyzer and saved for closer analysis.

Two basic scenarios, shown in Table 1, were designed. The intent of the Deploy scenario was to measure the performance of a single typical invocation. Accordingly, the invocation consisted of an Apache Axis deployment descriptor, which is used to initialize a service with Axis. The Stress scenario was designed to measure the performance of the system under a heavy load.

The protocols shown in Table 2 were intended mainly for testing the effects of compression and persistent connections. The single asynchronous protocol was only included as a rough start; it is not suitable for actual SOAP use, since it is unable to correlate different messages together without major changes, and is therefore incapable of proper two-way communication. None of these protocols (apart from HTTP) is really designed for serious use. However, their performance characteristics should be indicative of those of actual protocols with similar designs.

As can be seen from the descriptions, each new protocol is a minor modification of some other protocol. This is to maximize the chances that differences in the protocol performance numbers are actually the result of the changes made and not just random chance. For example, it was noticed that leaving the application-level message buffer too small for the actual message caused many-fold worsening in processing times; this was a clear difference between the needs of the compressing and the non-compressing protocols.

Table 2. The SOAP transfer protocols used in the tests

Protocol	Description
HTTP	The standard SOAP-over-HTTP binding shipped with Axis; synchronous request-response is implemented with HTTP's corresponding messages; HTTP 1.0 is used so a single connection is used only for a single interaction
PTCP	Messages consist of a content length in decimal followed by the actual SOAP content as XML with messages sent directly over a TCP connection; each connection is closed after a single request-response interaction; shipped with Axis for demonstration purposes
TCPZ	The PTCP protocol modified to compress each message's content with Gzip prior to sending
Pers	The PTCP protocol modified to keep connections open to permit several interactions to be carried out over a single TCP connection
Perz	The Pers protocol modified to compress each message's content with Gzip prior to sending
Bper	The Pers protocol modified to compress each message's content with an XML tokenizer prior to sending
Pera	The Pers protocol modified not to send response messages to permit the client to continue processing immediately without needing to wait for the server's response

The tokenizer used for Bper is a simple binary format inspired by WAP Binary XML [11]. It represents each XML element and attribute name as binary values, caching new names for later use. Standard element and attribute names used in SOAP messages are pre-cached. Integers, booleans, and dates are represented in a compact binary form, where the length of an integer's or a date's representation depends on its size.

The implementation of the tokenizing in Bper is quite suboptimal. Due to unfamiliarity with the Axis implementation at the time, it had to generate the binary representation from the message's XML representation. With the tokenizer implementation this necessitated parsing the XML at the sending end in addition to the ordinary parsing done at the receiving end. Therefore the main interest in the Bper protocol is the amount of data sent, and not the execution time.

The main intent of the experimentation was to measure SOAP's suitability for wireless environments. Therefore the Stress scenario, where differences should show more markedly, was run over four different connections, as described in Table 3, to measure the effect the network's degradation would have. The precise configuration of each connection is shown in Fig. 1. The Wireless LAN card used was Nokia D211 operating in ad hoc mode. Round trip times for the connections were measured with the ping program, i.e. ICMP Echo packets, and are given in minimum/average/maximum format, or as a single number where there was no variation.

Table 3. The connections used in the Stress scenario

Connection	Description
Localhost (LH)	Client and server on the same machine, communicating over the localhost network interface; round trip time 0.0 ms
LAN	Client and server on different machines, both connected to the same LAN; round trip time 0.1 ms
WLAN	Client and server on different machines, both connected to the same Wireless LAN; round trip time 2.3/2.5/4.4 ms
Routed WLAN (RW)	Client and server on different machines with a five-hop network route between them, the first hop of which is a Wireless LAN link; round trip time 9.2/13.8/25.8 ms

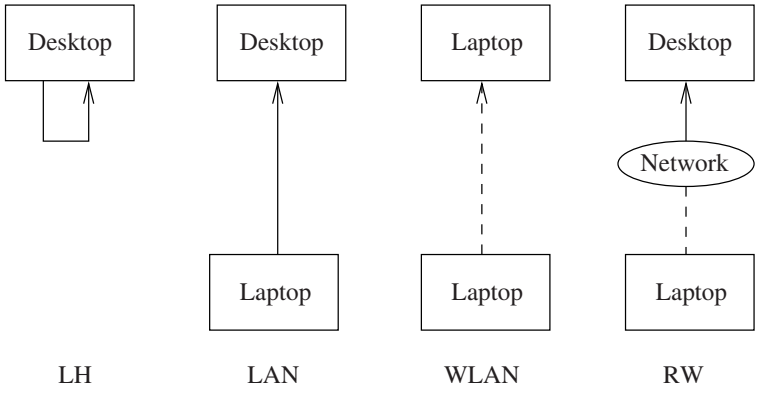

Fig. 1. Test configuration for the different connections; solid lines indicate localhost and wired connections, dashed lines indicate wireless connections

3 Test Results

In both scenarios, measurements were made of the amount of data each tested protocol sent over the network. The measured amount was further split into pure TCP packets (typically containing SYN, FIN, or ACK) and packets containing actual data. The count for data packets includes the IP and TCP headers. Total execution time was measured in the Stress scenario at the application level on the server side over all four tested connections. There was no noticeable fluctuation in the amount of data sent with different connections.

The amounts of data sent in the Deploy scenario are shown in Table 4. There is more variation in the results than would be expected, especially concerning the amount of pure TCP packets. This is explained by noting that the total number of packets differs among protocols, mostly depending on how much application-level buffering each protocol does. The amount of data sent by the Pera protocol is low only because it does not send a response message; if it did, its amount of data should be approximately equal to that of PTCP and Pers.

Table 4. The amount of data (in bytes) sent by the tested protocols in the single-request-response Deploy scenario

Protocol	Pure TCP	Data packets	Total data
HTTP	628	1399	2027
PTCP	628	1160	1788
TCPZ	560	678	1238
Pers	684	1165	1849
Perz	628	681	1309
Bper	560	583	1143
Pera	628	796	1424

Table 5. The amount of data (in bytes) sent by the tested protocols in the Stress scenario

Protocol	Pure TCP	Data packets	Total data
HTTP	10969736	27365474	38335210
PTCP	10967868	23138277	34106145
TCPZ	9773372	12566356	22339728
Pers	3824	21018279	21022103
Perz	4232	12655943	12660175
Bper	5592	5206834	5212426
Pera	95964	11528744	11624708

The amount of actual data sent is what would be expected. HTTP sends more data than the other protocols, since it has more extensive message headers. A protocol usable in practice would definitely require more headers than the simplistic message length included in PTCP (and by extension, all the others).

The effect of compression is relatively minor: Gzip compresses by approximately 40% and the binary representation by 50%. When taking also in the account the proportion pure TCP data takes of the communication, the compression does not have much advantage in this case.

The amounts of data sent in the Stress scenario are shown in Table 5. As mentioned earlier, these amounts remained effectively static independent of the connection used, so these numbers apply to all connections.

In this case the advantages of persistent connections are clearly visible, with the persistent protocols piggybacking almost all of their TCP ACKs in data packets. This permits the uncompressing Pers protocol to achieve a lower amount of total sent data than even the compressing TCPZ.

Again, as before, the amount of data sent by the Pera protocol is somewhat misleading. By not sending response messages, it manages to cut down the amount of sent data by half without any compression. If there were response messages, its amount in data packets should approach that of PTCP and Pers.

Table 6. Average execution times in milliseconds with mean deviations measured in the Stress scenario over the Localhost and LAN connections

Protocol	LH Total	LH Single	LAN Total	LAN Single
HTTP	130336±0.09%	7.460	145049±1.06%	8.302
PTCP	106698±0.13%	6.107	155599±0.66%	8.906
TCPZ	158209±0.11%	9.055	204916±0.67%	11.728
Pers	92819±0.12%	5.312	128290±0.12%	7.343
Perz	142039±0.21%	8.130	198882±0.10%	11.383
Bper	190944±1.35%	10.929	265647±0.26%	15.204
Pera	40556±0.87%	2.321	36178±2.82%	2.071

The messages sent in this scenario are only slightly larger than the ones in the Deploy scenario, so the performance of Gzip is also only slightly better, with compressed messages being approximately 50% of the size of the originals.

The binary representation's utilization of XML item caching permits it to compress messages to under 25% of their original size in this kind of repetitive scenario. While in this case the messages are highly similar, there would also be similarities in attributes and SOAP headers in more general cases, which should still permit this representation to have an advantage over Gzipped XML.

The XML item caching is not without its drawbacks, though. It requires that no messages are lost or reordered during transit, and that both ends can keep their caches synchronized, so for example application restarts are not acceptable. In this test scenario the reliability in messaging is achieved through TCP and a non-mobile terminal, and the simplicity and single purpose of the test applications preclude the possibility of caches falling out of synchrony. When the client is mobile, however, messages are typically lost during an address-changing handover. In addition, it is possible that the application protocol on top of the transport layer does not feed the messages upward sequentially. Therefore the protocol would need to ensure neither of these happens in actual situations.

The measured execution times, both total time taken and average time per invocation, for the Localhost and LAN connections are shown in Table 6. There is not much difference between these two connection types, and from the deviations it can be seen that there is little variation in measured times from one execution to another. The only noteworthy issue is that the execution time for the Pera protocol does not increase, indicating that the network is not its bottleneck at these bandwidths.

The times for the other two connections, WLAN and Routed WLAN, are shown in Table 7, for both total time and average per-invocation time as before. The measurements show considerable fluctuation depending on when measurements were made. From the deviations it can be seen that the times for the persistent protocols fluctuate wildly. This is an artifact of the Wireless LAN; its performance characteristics varied quite a lot depending on the measurement time.

The degradation in performance of the non-persistent protocols is particularly striking, which is even better visible in Fig. 2 summarizing the average

Table 7. Average execution times in milliseconds with mean deviations measured in the Stress scenario over the WLAN and Routed WLAN connections

Protocol	WLAN Total	WLAN Single	RW Total	RW Single
HTTP	509549±24.07%	29.164	2126201± 1.30%	121.692
PTCP	457714±10.95%	26.197	2253455± 4.01%	128.975
TCPZ	475750±13.96%	27.229	2195746± 2.56%	125.672
Pers	255564± 6.02%	14.627	393324±42.54%	22.511
Perz	417271±26.04%	23.882	391899±33.86%	22.430
Bper	415225± 6.62%	23.765	416045±27.32%	23.812
Pera	81396± 1.46%	4.659	113035±20.68%	6.469

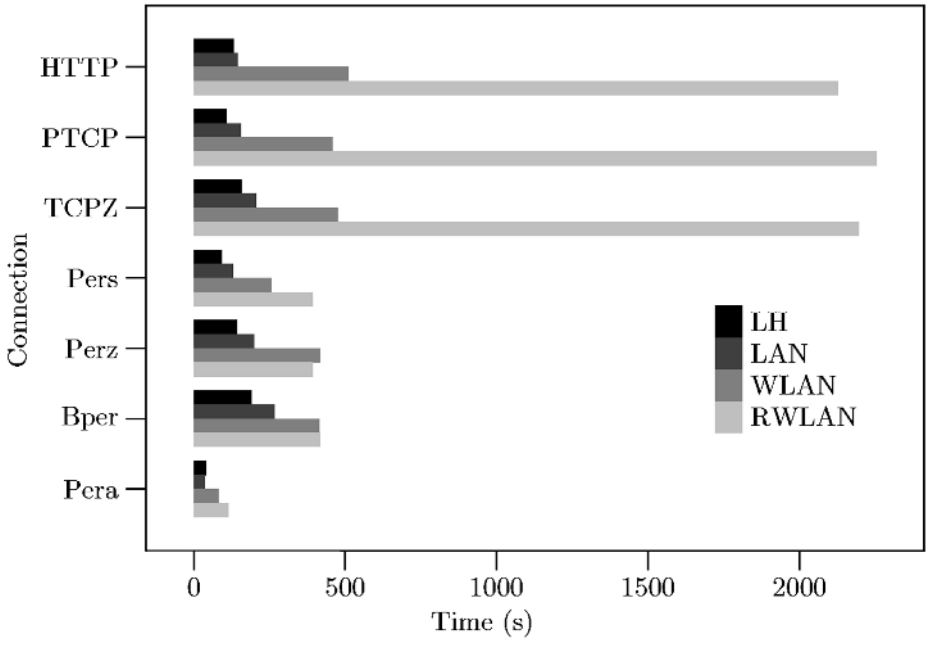

Fig. 2. Average times for all protocols and connections

times for all protocols and connections. The main cause of this degradation is the fickle nature of the Wireless LAN, which proved to be quite willing to drop ACKs in response to connection-opening SYNs, causing several-second delays as the SYN sender's timeout triggered.

The performance of the compressing persistent protocols can even be seen to improve when moving from the WLAN connection to the Routed WLAN connection. This is most probably an artifact of the fickleness of the Wireless LAN. However, since the performance of the compressing protocols in Routed

WLAN is similar to that of Pers, it would seem to imply that the time taken for compression is not a significant factor in the runtime anymore.

4 Conclusions

From the measurements we can see that the default way of using SOAP is significantly suboptimal, especially under heavy communication load in slower networks. In the test scenarios the main problem was the need to open new connections for new invocations, mainly because Wireless LAN bandwidth was not a bottleneck when invocations were synchronous. However, the effect of bandwidth reduction is clearly visible with the asynchronous Pera protocol.

The generic Gzip compression algorithm manages to decrease message size only by about 50%. Observations made during testing reveal that compressing and decompressing also consume significant amounts of processor time, which is directly visible in measured execution times on Localhost and LAN. In the Stress scenario the binary format manages to decrease message size by approximately 75%.

The processor time used is especially significant when considering the power of devices typically having wireless links, such as phones or PDAs. Preliminary experimentation seems to indicate that the extra processing time incurred by binary compression can be mostly eliminated by directly generating the compressed format, as would be expected. Furthermore, significant speedups have been achieved by parsing a binary XML format directly without going through the textual form [7].

Keeping connections open is clearly a benefit in the Stress scenario due to the heavy load the scenario places on the network. The effects become more pronounced with increased connection latency and unreliability, as would be expected. Persistence of connections would also benefit interactive applications communicating with a small number of servers, since it eliminates the round trip needed in TCP connection establishment. This could, and in the future probably will, also be achieved by using HTTP 1.1 with its built-in persistent connections.

The performance of the Pera protocol shows that further reducing the effects of latency is possible for applications able to send several messages before needing possible responses to any of them. This could include e.g. event-based systems, where no application-level response is returned to the event generator. However, this is more of an issue for the programming framework and not the protocol, since an asynchronous interface can be built on top of a synchronous protocol if needed.

The latency becomes even more of a problem when considering wireless technology used in mobile phones, such as GPRS. Measurements show that the average latency for a GPRS-based connection equivalent to the Routed WLAN is between 800 and 900 ms with large variations. This kind of connection would make it completely unsuitable to use a synchronous interface if it could at all be avoided. A GPRS connection also has a much lower bandwidth than Wireless LAN. Preliminary experiments would seem to indicate that the Stress scenario

with the Pera protocol would take over an hour to run with a GPRS-based connection.

Future plans are to extend these tests to also cover a GPRS-based connection. For these tests it would also be useful to have a new protocol designed to implement the optimizations shown most beneficial by the above results. The future tests are also expected to cover a new scenario based on some real-world situation with a heavy network load.

References

1. Elan Amir, Hari Balakrishnan, Srinivasan Seshan, Randy Katz. Efficient TCP over networks with wireless links. In *Proceedings of the Fifth IEEE Workshop of Hot Topics in Operating Systems*, May 1995.
2. Olivier Avaro, Philippe Salembier. MPEG-7 Systems: Overview, *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):760–764, June 2001.
3. Hari Balakrishnan, Venkata Padmanabhan, Srinivasan Seshan, Randy Katz. A comparison of mechanisms for improving TCP performance over wireless links, *IEEE/ACM Transactions on Networking*, 5(6):756–769, December 1997.
4. Tim Berners-Lee, Roy Fielding, Henrik Frystyk Nielsen. *RFC 1945: Hypertext Transfer Protocol – HTTP/1.0*, May 1996. `http://www.ietf.org/rfc/rfc1945.txt`
5. Antaeus Feldspar. *An Explanation of the DEFLATE Algorithm*, August 1997. `http://www.gzip.org/deflate.html`
6. Roy Fielding, James Gettys, Jeffrey Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul Leach, Tim Berners-Lee. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*, June 1999. `http://www.ietf.org/rfc/rfc2616.txt`
7. Marc Girardot, Neel Sundaresan. Millau: an encoding format for efficient representation and exchange of XML over the Web. In *Ninth International World Wide Web Conference*, May 2000. `http://www9.org/w9cdrom/154/154.html`
8. Hartmut Liefke, Dan Suciu. XMill: an efficient compressor for XML data. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, May 2000.
9. World Wide Web Consortium (W3C). *W3C Note: Simple Object Access Protocol (SOAP) 1.1*, May 2000. `http://www.w3.org/TR/SOAP/`
10. World Wide Web Consortium (W3C). *W3C Proposed Recommendation: SOAP Version 1.2 Part 1: Messaging Framework and SOAP Version 1.2 Part 2: Adjuncts*, May 2003. `http://www.w3.org/TR/2003/PR-soap12-part1-20030507/` and `http://www.w3.org/TR/2003/PR-soap12-part2-20030507/`
11. World Wide Web Consortium (W3C). *W3C Note: WAP Binary XML Content Format*, June 1999. `http://www.w3.org/TR/wbxml/`